

**EV369764814**

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

**APPLICATION FOR LETTERS PATENT**

**Relational Database Schema Version Management**

Inventors:

Leela S. Tamma  
Krishna Vitaldevara

ATTORNEY'S DOCKET NO. MS1-1880US

## **TECHNICAL FIELD**

[0001] This invention relates to relational database design and development, and more specifically to a methodology for managing versions of a relational database schema.

## **BACKGROUND**

[0002] In software development, specifically relational database applications, maintaining a relational database schema through multiple releases of the database can be very cumbersome.

[0003] When a new version of a database is developed, prior to release of the database, an installation file associated with the new version must be created, which, when executed, will install from scratch, the new version of the database. In addition to an installation file, an upgrade file must be created that, when executed, will upgrade a previous version of the database to the new version of the database. Even more complicated is a scenario in which more than one previous version of the database exists. In this case, a different upgrade file is required to upgrade from each previous version of the database to the current version of the database. Furthermore, each installation file and upgrade file must be tested to insure that they all result in the same database structure.

[0004] Database installation files and upgrade files are typically generated manually by a database developer who is familiar with the intricacies of the database schema. This is a very time-consuming and error-prone method.

[0005] Accordingly, a need exists for a technique that enables automated generation of relational database installation and upgrade files.

## **SUMMARY**

[0006] A technique for managing multiple versions of a relational database schema is described. Schema data associated with multiple versions of a relational database are maintained according to a database schema version management structure. The schema data identifies, for a particular version, data definition language (DDL) scripts, data manipulation scripts (DML), and drop scripts that are to be applied to upgrade from the previous version of the database to the particular version of the database.

[0007] Laws of set theory are applied to the database schema data to identify which DDL scripts and DML scripts are to be executed in order to create a full install of a particular version of the database. Similarly, laws of set theory are applied to the database schema data to identify which DDL scripts, DML scripts, and drop scripts are to be executed in order to upgrade from a given version of the database to a particular newer version of the database.

[0008] The identified scripts are copied into an installation file (or upgrade file), which can then be executed to perform the appropriate install (or upgrade).

## **BRIEF DESCRIPTION OF THE DRAWINGS**

[0009] Figure 1 is a block diagram that illustrates exemplary schemas associated with four sequential versions of a relational database.

[0010] Figure 2 is an XML schema definition (XSD) that defines the structure of an exemplary XML file that may be used to describe the schemas of multiple sequential versions of a relational database.

[0011] Figure 3 is an XML file structured according to the XSD illustrated in Figure 2 that describes the schemas of the four sequential versions of the relational database illustrated in Figure 1.

[0012] Figure 4 is a block diagram that illustrates in set theory notation, an exemplary method for generating an installation file for creating an initial version of a relational database.

[0013] Figure 5a is a block diagram that illustrates in set theory notation, an exemplary method for generating an installation file for creating a full install of a non-initial version of a relational database.

[0014] Figure 5b is a block diagram that illustrates in set theory notation, an exemplary method for determining a set of data definition language scripts to be included in a relational database installation file.

[0015] Figure 5c is a block diagram that illustrates in set theory notation, an exemplary method for determining a set of data manipulation language scripts to be included in a relational database installation file.

[0016] Figure 6a is a block diagram that illustrates in set theory notation, an exemplary method for generating an upgrade file for upgrading a relational database from one version to a newer version.

[0017] Figure 6b is a block diagram that illustrates in set theory notation, an exemplary method for generating a set of data definition language scripts associated with a relational database upgrade file.

[0018] Figure 6c is a block diagram that illustrates in set theory notation, an exemplary method for generating a set of data manipulation language scripts associated with a relational database upgrade file for modifying data manipulation language objects.

**[0019]** Figure 6d is a block diagram that illustrates in set theory notation, an exemplary method for generating a set of data manipulation language scripts associated with a relational database upgrade file for creating data manipulation language objects.

**[0020]** Figure 6e is a block diagram that illustrates in set theory notation, an exemplary method for generating a set of data manipulation language scripts associated with a relational database upgrade file for dropping data manipulation language objects.

**[0021]** Figure 7 is a block diagram that illustrates in set theory notation, an exemplary method for generating a set of data manipulation language scripts associated with data manipulation language objects that are created or modified in an upgrade of a relational database.

**[0022]** Figure 8 is a block diagram that illustrates select components of an exemplary computer system in which a database schema version management system may be implemented.

**[0023]** Figure 9 is a block diagram that illustrates an exemplary environment in which a database schema version management system (DSVMS) may be implemented.

**[0024]** Figure 10 is a flow diagram that illustrates an exemplary method for installing an initial version of a relational database.

**[0025]** Figure 11 is a flow diagram that illustrates an exemplary method for installing a non-initial version of a relational database.

**[0026]** Figure 12 is a flow diagram that illustrates an exemplary method for upgrading a relational database.

## **DETAILED DESCRIPTION**

### **Overview**

[0027] The embodiments described below provide techniques for managing schema data associated with multiple sequential versions of a relational database such that script files can be automatically generated to create either a full install of any version of the relational database or an upgrade from one version of the database to any newer version of the database. In the described exemplary implementation, an XML schema definition (XSD) defines a structure for an XML file that is used to store metadata associated with the schemas of multiple sequential versions of a relational database. The metadata describes data definition language (DDL) files and data manipulation language (DML) files that are associated with each version of the relational database. The metadata for each version of the relational database is grouped into three categories: (1) DDL files, which include scripts to create, drop, or modify database objects such as tables, indexes, views, and so on; (2) create/alter DML files, which include scripts to create or modify data manipulation objects such as functions and stored procedures that are new or have changed from one version to another; and (3) drop DML files, which include scripts to drop data manipulation objects that were created in support of a previous version of the relation database, but that are no longer needed.

[0028] In the described exemplary implementation, laws of set theory are applied to the schema data that is maintained to generate a script file that contains the commands used to fully install a particular version of the relational database from scratch. Alternatively, laws of set theory may be applied to the schema data

that is maintained to generate a script file that contains the commands used to upgrade an installed version of the database to a newer version of the database.

### **Relational Database Schema Version Management**

[0029] Figure 1 illustrates exemplary schema data associated with four sequential versions of a relational database. In the illustrated example, box 102 represents version 1.0, which is an initial release of a relational database. Version 1.0 of the database includes an employee table, a customer table, three system messages (i.e., SysMessage1, SysMessage2, and SysMessage3), two logins (i.e., login1 and login2), two procedures (i.e., procedures A and B), and two functions (i.e., functions A and B).

[0030] Box 104 represents a second version (i.e., version 1.1) of the same relational database. Changes that occurred between version 1.0 and version 1.1 include modifications to the customer table, creation of a products table, creation of a purchases view, modification of procedure A, creation of new procedure C, deletion of function A, and creation of new function C.

[0031] Box 106 represents a third version (i.e., version 1.2) of the same relational database. Changes that occurred between version 1.1 and version 1.2 include deletion of the purchases view, deletion of SysMessage2, modifications to SysMessage3, creation of SysMessage4, modifications to procedure B, and creation of procedure D.

[0032] Box 108 represents a fourth version (i.e., version 3.0) of the same relational database. Changes that occurred between version 1.2 and version 3.0 include modifications to the employee table, deletion of login2, creation of login3, modifications to procedure A, deletion of procedure B, creation of procedure E, creation of procedure F, and modifications to function C.

[0033] It is clear from the illustration in Figure 1 that supporting users of multiple versions of the illustrated database can be very cumbersome as the users upgrade from one version to another. For example, when version 1.2 is released, new users will install version 1.2 of the database from scratch. In this scenario, an installation file for installing version 1.2 of the relational database accounts for database objects created in version 1.0, version 1.1, and version 1.2, as well as any modifications to objects performed when upgrading to version 1.1 or version 1.2, as well as any deletes performed when upgrading to version 1.1 or version 1.2 are included in.

[0034] Similarly, different upgrade files can be generated to upgrade existing users from version 1.1 to version 1.2 or to upgrade existing users from version 1.0 to version 1.2. That is, a file to upgrade an existing user from version 1.1 to version 1.2 accounts for changes to the database structure that occurred between version 1.1 and version 1.2. On the other hand, a script to upgrade an existing user from version 1.0 to version 1.2 accounts for changes to the database structure that occurred between versions 1.0 and 1.1 and changes to the database structure that occurred between versions 1.1 and 1.2.

### **Database Schema Version Management XML Schema Definition**

[0035] Figure 2 illustrates an XML schema definition (XSD) 202 that defines the structure of an exemplary XML file that may be used to maintain schema data associated with multiple sequential versions of a relational database. An XML file structured according to XSD 202 includes a single element 204 named “DSVM” (database schema version management) of type “DSVMType”.

[0036] Type definition 206 specifies the structure of an element with type equal to “DSVMType”. As specified by type definition 206, any element having type

“DSVMType” is made up of a sequence of “Release” elements, each of type “ReleaseType”. The sequence may have any number of “Release” elements (as indicated by the maxOccurs=“unbounded” specification, but must have at least one “Release” element (as indicated by the minOccurs=“1” specification).

[0037] Type definition 208 specifies the structure of an element with type equal to “ReleaseType”. As specified by type definition 208, any element having type “ReleaseType” has a version attribute 210 and a sequence 212 of three elements, “ddl”, “sproc”, and “drop”, each of type “SchemaDataType”. The sequence 212 may include zero or more of each of the specified elements. In the described exemplary implementation, a ddl element is used to specify a file containing one or more scripts to create, modify, or delete data definition language (DDL) objects (e.g., tables, views, system messages, logins, etc.); a sproc element is used to specify a file containing a data manipulation language (DML) script (e.g., stored procedures, functions, etc.); and a drop element is used to specify a file containing a script to drop a data manipulation language (DML) object.

[0038] Type definition 214 specifies the structure of an element with type equal to “SchemaDataType” (i.e., ddl, sproc, and drop elements of a ReleaseType element).

[0039] Header 216 includes attributes that identify a namespace associated with the schema. The targetNamespace attribute makes it possible to reuse schema definitions and declarations between schemas. The header as illustrated is well-known to those skilled in the art.

### **XML-Based Management of Relational Database Schema Version Data**

[0040] Figure 3 illustrates an exemplary XML file 302 structured according to XSD 202 illustrated in Figure 2. XML file 302 describes the schemas of relational

database versions 102, 104, 106, and 108 as illustrated in Figure 1. In the described exemplary implementation, XML file 302 is created and managed by an individual (e.g., the designer and/or developer of the relational database that is being represented). For each release of the relational database, a release element that includes a sequence of entries is added to XML file 302 to specify script files that, when executed, will upgrade the previous version of the database to the current version of the database. For example, if there are entries in an XML file that correspond to versions 1.0, 1.1, 1.2, and 3.0 of a relational database, then the entries associated with version 1.1 specify the changes between version 1.0 and version 1.1. Similarly, the entries associated with version 1.2 specify the changes between version 1.1 and version 1.2; and the entries associated with version 3.0 specify the changes between version 1.2 and 3.0.

**[0041]** In addition to being grouped by version, references to script files in XML file 302 are also presented in the order in which they are to be executed. For example, if two functions and a stored procedure are being added for a particular version, if the stored procedure references the two functions, then the two functions need to be created before the stored procedure is created. Accordingly, entries associated with the two functions are listed in the XML file before an entry associated with the stored procedure.

**[0042]** Release elements 304, 306, 308, and 310 correspond to versions 1.0, 1.1, 1.2, and 3.0, respectively, of the relational database, as illustrated in Figure 1. The version attribute for each release element is assigned a sequential number that corresponds to the order in which the database versions were released. The value of the version attributes may correspond to the version number assigned to each release of the database, but, as illustrated in Figure 3, the value of the version

attribute associated with a particular release element does not necessarily correspond to the version number associated with the actual database release. Release element 304 has version attribute “1”, corresponds to relational database version 1.0, and includes ddl elements and sproc elements. Because version 1.0 is an initial release of the relational database, no elements need to be dropped from a previous version, so there are no drop elements specified. The first ddl element specifies a file named “Tables1.0.sql” that includes DDL scripts to create, for example, the Employee Table and the Customer Table. The second ddl element specifies a file named “SysMessages1.0.sql” that includes DDL scripts to create SysMessage1, SysMessage2, and SysMessage3. The third ddl element specifies a file named “Login1.0.sql” that includes DDL scripts to create Login1 and Login2.

**[0043]** The first sproc element specifies a file named “ProcA.sql” that includes a DML script associated with procedure A. The second sproc element specifies a file named “FuncA.sql” that includes a DML script associated with function A. The third sproc element specifies a file named “ProcB.sql” that includes a DML script associated with procedure B. The fourth sproc element specifies a file named “FuncB.sql” that includes a DML script associated with function B. In the described exemplary implementation, the scripts in the specified DML files do not include the key word that defines the operation to be performed, such as “create” or “alter”. These key words are prepended to scripts as needed, as will be described in further detail below.

**[0044]** Release element 306 corresponds to version 1.1 of the relational database and includes elements that represent changes to the database between version 1.0 and version 1.1. Release element 306 includes a ddl element, three sproc elements, and a drop element. The ddl element specifies a file named

“UpgradeTables1.1.sql” that includes ddl scripts to modify the customer table, create a products table, and create a purchases view (i.e., the DDL changes between version 1.0 and version 1.1). The sproc elements specify files named “ProcA.sql”, “ProcC.sql”, and “FuncC.sql”. The file named “ProcA.sql” contains a script associated with procedure A, which is to be modified; the file named “ProcC.sql” contains a script associated with procedure C, which is to be created; and the file “FuncC.sql” contains a script associated with function C, which is to be created. The drop element specifies a file named “FuncA.sql” that contains a script. When executed, the script causes function A, which exists in version 1.0, to be dropped.

**[0045]** Similarly, release element 308 corresponds to version 1.2 of the relational database and includes elements that represent changes to the database between version 1.1 and version 1.2. Release element 308 includes a ddl element and two sproc elements. The ddl element specifies a file named “UpgradeSysMessages1.2.sql” that includes ddl scripts that, when executed, cause existing purchase view and SysMessage2 to be dropped, existing SysMessage3 to be modified, and new SysMessage4 to be created. The two sproc elements specify files named “ProcB.sql” and “ProcD.sql”. Release element 308 does not include a drop element, indicating that there are no DML objects that were dropped between version 1.1 and version 1.2.

**[0046]** Finally, release element 310 corresponds to version 3.0 of the relational database and includes elements that represent changes to the database between version 1.2 and version 3.0. Release element 310 includes two ddl elements, four sproc elements, and one drop element. The ddl elements specify files named “UpgradeTables3.0.sql” and “UpgradeLogins3.0.sql” that include ddl scripts that,

when executed, cause the employee table to be modified, existing login2 to be dropped, and new login3 to be created. The sproc elements specify files named “ProcA.sql”, “FuncC.sql”, “ProcE.sql”, and “ProcF.sql”. As illustrated in Figure 1, procedure A and function C are modified, and procedure E and procedure F are added when moving from version 1.2 to version 3.0. The drop element specifies a file named “ProcB.sql” that contains a script that, when executed, causes procedure B to be dropped.

### **Installation File Generation for Initial Database Version**

[0047] Figure 4 illustrates in set theory notation, an exemplary method for generating an installation file for an initial version of a relational database. Set 402 includes DDL scripts for creating DDL objects associated with the initial version of the relational database. For version 1.0 of the example relational database represented in Figures 1 and 3, set 402 includes the scripts in files “Tables1.0.sql”, “SysMessages1.0.sql”, and “Login1.0.sql”.

[0048] Set 404 includes DML scripts (prepended with a “create” command) for creating DML objects associated with the initial version of the relational database. For version 1.0 of the example relational database represented in Figures 1 and 3, set 404 includes the scripts in files “ProcA.sql”, “FuncA.sql”, “ProcB.sql”, and “FuncB.sql” (each prepended with a “create” command”).

[0049] As indicated in Figure 4, the union of set 402 and set 404 results in a version 1.0 installation file for the relational database.

### **Installation File Generation for a Non-Initial Database Version**

[0050] Figure 5a illustrates in set theory notation, an exemplary method for generating an installation file for a non-initial version of a relational database. For example, if there have been versions 1.0, 1.1, and 1.2 of a relational database,

Figure 5 illustrates a method for generating an installation file for a new user that, when executed, will fully install all of the database elements that exist in version 1.2, without requiring the user to first install version 1.0, upgrade to version 1.1, and finally upgrade to version 1.2.

**[0051]** Installation file 502 for a non-initial version (version  $i$ ) of a relational database contains the union of sets 504 and 506. Set 504 includes DDL scripts from version  $i$  and previous versions of the database. For example, given the example relational database represented in Figures 1 and 3, for creation of version 1.1, set 504 includes the DDL scripts from versions 1.0 and 1.1; for creation of version 1.2, set 504 includes the DDL scripts from versions 1.0, 1.1, and 1.2.

**[0052]** Set 506 includes DML scripts from version  $i$  and previous versions of the database, not including scripts for those DML objects that are dropped and not re-created between the initial version and version  $i$  of the relational database. For example, given the example relational database represented in Figures 1 and 3, for creation of version 1.1, set 506 includes the DML scripts contained in files “Tables1.0.sql”, “SysMessages1.0.sql”, “Login1.0.sql”, “UpgradeTables1.1.sql”, “ProcA.sql”, “ProcB.sql”, “FuncB.sql”, “ProcC.sql”, “FuncC.sql”, but does not include scripts contained in the file “FuncA.sql” because “FuncA.sql” is specified as a drop element associated with version 1.1.

**[0053]** Figure 5b illustrates in set theory notation, an exemplary method for determining set 504. Set 504 is equal to the union of sets of DDL scripts for each version of the database, up to and including version  $i$ . That is, the union of sets 508, 510, ..., 512, where set 508 includes DDL scripts associated with the first version, set 510 includes DDL scripts associated with the second version, and

so on, up to set 512, which includes DDL scripts associated with the  $i^{\text{th}}$  version of the relational database.

**[0054]** Figure 5c illustrates in set theory notation, an exemplary method for determining set 506. Set 506 includes those scripts found in the union of sets 514 and 516 that are not also found in set 518, where set 514 includes DML scripts associated with version  $i-1$ , set 516 includes DML scripts associated with an upgrade from version  $i-1$  to version  $i$ , and set 518 includes drop scripts associated with an upgrade from version  $i-1$  to version  $i$ . As illustrated in Figure 5c, the process of generating set 506 is iterative. First, set 520 is generated to include DML scripts associated with a full install of the second version of a relational database. Set 520 includes DML scripts associated with the first version (set 522) and DML scripts associated with the second version (set 524), except for any scripts associated with DML objects that get dropped (set 526) when upgrading from the first version to the second version. Similarly, set 528 is generated, which includes DML scripts associated with a full install of the third version, and is mathematically expressed in Figure 5c as set 528 equal to set 530 union set 532 minus set 534. This process is continued as indicated in Figure 5c until set 506 is determined.

### **Database Upgrade File Generation**

**[0055]** Figure 6a illustrates in set theory notation, an exemplary method for generating an upgrade file for upgrading from one version to another version of a relational database. For example, if there have been versions 1.0, 1.1, and 1.2 of a relational database, Figure 6a illustrates a method for generating an upgrade file that, when executed, will upgrade an existing version 1.0 instance of the relational

database to version 1.2 of the relational database, without requiring the user to first upgrade to version 1.1 and then upgrade to version 1.2.

[0056] Upgrade file 602 for upgrading from version  $j$  to version  $k$  of a relational database includes scripts represented by sets 604, 606, 608, and 610 (shown with the union symbol between them in Figure 6a). Set 604 includes the DDL scripts for upgrading from version  $j$  to version  $k$ . Set 606 includes DML scripts associated with DML objects that exist in version  $j$  and get modified between version  $j$  and version  $k$ . Set 608 includes DML scripts associated with DML objects that exist in version  $k$ , but that don't exist in version  $j$ . Set 610 includes drop scripts associated with DML objects that exist in version  $j$ , but that don't exist in version  $k$ .

[0057] Figure 6b illustrates in set theory notation, an exemplary method for determining set 604. Set 604 is the union of sets 612, 614, ..., 616, where set 612 represents DDL scripts for upgrading from version  $j$  to version  $j+1$ ; set 614 represents DDL scripts for upgrading from version  $j+1$  to version  $j+2$ ; and so on up to set 616, which represents DDL scripts for upgrading from version  $k-1$  to version  $k$ .

[0058] Figure 6c illustrates in set theory notation, an exemplary method for determining set 606. Set 606 includes DML scripts that alter DML objects that are modified between version  $j$  and version  $k$  of a relational database. Set 606 is illustrated in Figure 6 as being the intersection of sets 618 and 620, where set 618 is a set of scripts for DML objects that exist in version  $j$  and set 620 is a set of scripts for DML objects that are created or modified between version  $j$  and version  $k$  of a relational database. Determining set 618 is described above with

reference to determining set 506 as illustrated in Figure 5c. Determination of set 620 is described in further detail below with reference to Figure 7.

**[0059]** Figure 6d illustrates in set theory notation, an exemplary method for determining set 608. Set 608 includes DML scripts associated with new DML objects that are created between version  $j$  and version  $k$  of a relational database. Set 608 is illustrated in Figure 6d as being the difference between sets 620 and 606, where set 620, as described above, is a set of scripts for DML objects that are created or modified between version  $j$  and version  $k$ ; and set 606, as described above, is a set of scripts that modify DML objects that exist in version  $j$  and get modified between version  $j$  and version  $k$ .

**[0060]** Figure 6e illustrates in set theory notation, an exemplary method for determining set 610. Set 610 includes drop scripts for DML objects that are dropped and not re-created between version  $j$  and version  $k$ . Set 610 is illustrated in Figure 6e as being the difference between (A) the union of sets 622, 624, ..., 626 and (B) set 620, where sets 622, 624, ..., 626 represent drop scripts associated with versions  $j+1, j+2, \dots, k$ , respectively; and where set 620, as described above, is a set of scripts for DML objects that are created or modified between version  $j$  and version  $k$ .

**[0061]** Figure 7 illustrates in set theory notation, an exemplary method for determining set 620, which is a set of scripts for DML objects that are created or modified between version  $j$  and version  $k$  of a relational database. Determination of set 620 is an iterative process, and is very similar to the process described above for determining set 506, as illustrated in Figure 5c. First, set 702 is determined as a set of DML scripts that create or modify DML objects between version  $j$  and version  $j+1$ . These are represented as set 704, which includes scripts that are

specified as sproc elements associated with version  $j+1$  in a database schema version management (DSVM) XML file associated with the relational database, such as XML file 302 shown in Figure 3.

[0062] Next, set 706 is determined as a set of DML scripts that create or modify DML objects between version  $j$  and version  $j+2$ . As illustrated in Figure 7, set 706 includes those elements found in the union of sets 702 and 708 that are not also found in set 710, where set 702, as described above, is a set of DML scripts that create or modify DML objects between version  $j$  and version  $j+1$ ; set 708 is a set of DML scripts that create or modify DML objects between version  $j+1$  and version  $j+2$ ; and set 710 is a set of drop scripts associated with DML objects that are dropped between version  $j+1$  and version  $j+2$ . The drop scripts of set 710 are specified as drop elements associated with version  $j+2$  in a DSVM XML file associated with a relational database.

[0063] Similarly, set 712 is a set of DML scripts that create or modify DML objects between version  $j$  and version  $j+3$ . As illustrated in Figure 7, set 712 includes those elements found in the union of sets 706 and 714 that are not also found in set 716, where set 706, as described above, is a set of DML scripts that create or modify DML objects between version  $j$  and version  $j+2$ ; set 714 is a set of DML scripts that create or modify DML objects between version  $j+2$  and version  $j+3$ ; and set 716 is a set of drop scripts associated with DML objects that are dropped between version  $j+2$  and version  $j+3$ .

[0064] This process continues until, finally, set 620 is generated so as to include those elements found in the union of sets 718 and 720 that are not also found in set 722, where set 718 is a set of DML scripts that create or modify DML objects between version  $j$  and version  $k-1$ ; set 720 is a set of DML scripts that

create or modify DML objects between version  $k-1$  and version  $k$ ; and set 722 is a set of drop scripts associated with DML objects that are dropped between version  $k-1$  and version  $k$ .

### **Database Schema Version Management System**

[0065] Figure 8 illustrates select components of an exemplary computer system 802 in which a database schema version management system as described herein may be implemented. Computer system 802 includes a processor 804, a memory 806, and input/output interfaces 808. Input/output interfaces 808 provide a means by which computer system 802 can communicate with other devices, including, but not limited to, other computers, a keyboard, a mouse, and a display device. Operating system 810 is stored in memory 806 and executed on processor 804 to provide a software platform on which applications 812 may run. Applications 812 are also stored in memory 806 and executed on processor 804. Examples of applications 812 may include, but are not limited to, word processing applications, spreadsheet applications, user interface development applications, database development applications, and so on.

[0066] Database schema version management system (DSVMS) 814 is a specific application that is stored in memory 806 and executed on processor 804. DSVMS 814 maintains metadata associated with the schemas of multiple versions of a relational database and uses the schema metadata to generate installation or upgrade files for versions of the database.

[0067] Figure 9 illustrates select components of an exemplary DSVMS 814, illustrated within an exemplary software development environment 900. DSVMS 814 includes database schema version management XML schema definition (DSVM XSD) 902, DSVM XML file 904, data definition language

(DDL) scripts 906, data manipulation language (DML) scripts 908, and upgrade/installation file generator 910

**[0068]** XSD 902 is an XML schema definition that defines a structure for XML file 904. An example DSVM XSD is described above with reference to Figure 2. XML file 904 is structured according to DSVM XSD 902, and maintains metadata associated with the schemas of one or more sequential versions of a relational database. An example DSVM XML file is described above with reference to Figure 3.

**[0069]** DDL scripts 906 are scripts that can be run to create, modify, or drop DDL objects (e.g., tables, views, logins, system messages, etc.) of a relational database. DML scripts 908 are scripts that can be run to create, modify, or drop DML objects (e.g., stored procedures, functions, etc.) of a relational database.

**[0070]** Upgrade/installation file generator 910 is configured to assemble a file of scripts that, when executed, will install a particular version of a relational database, or upgrade an existing installation of the database from one version to another. In an exemplary implementation, upgrade/installation file generator 910 applies laws of set theory (e.g., as described above with reference to Figures 4-7) to database schema metadata stored in DSVM XML file 904. The results of the set theory calculations are then used to identify specific DDL scripts 906 and DML scripts 908 that are to be included in a particular upgrade or installation file. In an exemplary implementation, upgrade/installation file generator 910 is implemented using C#. However, it is recognized that any number of programming languages may be used to implement upgrade/installation file generator 910 including, but not limited to, C, C++, or Java.

**[0071]** Environment 900 is an exemplary database application development environment in which an exemplary DSVMS 814 may be implemented. Environment 900 includes a development server 912, and may also include a testing server 914 and a production server 916. Development server 912 includes a development version of relational database 918, and may also include a development version of an application 920, which is configured, for example, to access the development version of database 918. Testing server 914 includes a testing version of relational database 918 and a testing version of application 920, which is configured, for example, to access the testing version of database 918. Similarly, production server 916 includes a production version of relational database 918 and a production version of application 920.

**[0072]** In an exemplary implementation, development server 912 is used by software developers to create upgrades to relational database 918 and application 920. Testing server 914 is configured to mimic a production environment to enable thorough testing of a new version of relational database 918 and application 920, before the new version is ported to production server 916, where it is available for real-world operation.

**[0073]** DSVMS 814 may be installed on development server 912 or may be installed on a separate computer system not illustrated in Figure 9. Alternatively, different portions of DSVMS 814 may reside on different physical computer systems. For example, upgrade/installation file generator 910 may reside on a separate computer system (not illustrated in Figure 9) while DSVM XSD 902, DSVM XML 904, DDL scripts 906, and DML scripts 908 may reside on development server 912.

[0074] In an exemplary implementation, after the release of a particular version, development server 912, testing server 914, and production server 916 all house the same version of application 920 and relational database 918. When development of a new version begins, the version housed on testing server 914 and production server 916 does not change, but a new version being developed is housed on development server 912. As a new database version is developed, version-specific DDL and DML scripts are developed and/or version-specific modifications are made to existing DDL and DML scripts. Furthermore, version-specific database schema metadata is stored in DSVM XML to record a description of changes that are made from the previous version to the new version.

[0075] After database development is complete for the new version, upgrade/installation file generator 910 is invoked to generate an upgrade file to upgrade the test version of relational database 918 stored on testing server 914 to the new version of the database, which corresponds to the new version stored on development server 912.

[0076] Similarly, after testing is complete, upgrade/installation file generator 910 may be invoked to generate an upgrade file (or an installation file for new users) to upgrade relational database 918 stored on production server 916 to the new version.

#### **Initial Version Database Installation Method**

[0077] Figure 10 is a flow diagram that illustrates an exemplary method 1000 for installing an initial version of a relational database. The illustrated process can be implemented in any suitable hardware, software, firmware or combination thereof.

**[0078]** At block 1002, the system copies scripts associated with data definition language (DDL) objects of the initial version of the database into an installation file. For example, upgrade/installation file generator 910 examines data in XML file 904 to identify files containing DDL scripts associated with the first version of the database. In the described exemplary implementation, the DDL script files are identified by the ddl elements associated with the first release element in the XML file. The contents of the identified files are then copied into an installation file.

**[0079]** At block 1004, the system prepends a “create” command to data manipulation language (DML) scripts associated with the initial version of the database, and copies the scripts into the installation file. For example, upgrade/installation file generator 910 examines data in XML file 904 to identify files containing DML scripts associated with the first version of the database. In the described exemplary implementation, the DML script files are identified by the sproc elements associated with the first release element in the XML file. In an exemplary implementation, the DML script files do not include a keyword operator (“create”, “alter”, “drop”, etc.), so a “create” command is prepended to the scripts as they are copied into the installation file.

**[0080]** The installation file that is generated contains scripts to create DDL and DML objects associated with the initial version of the database, and running the installation file will cause a new instance of the initial version of the database to be created. Blocks 1002 and 1004 of method 1000 are also illustrated in set theory notation in Figure 4, and described with reference thereto above.

**[0081]** In the described implementation, there is a DML script file for each DML object associated with the database. If a particular function or stored procedure, for example, changes from version 1.0 to version 1.1, the script file is

changed. Therefore, there is no way to recreate version 1.0 after version 1.1 has been released.

**[0082]** In an alternate implementation, however, DML scripts may be stored in different subdirectories, each corresponding to a different release of the database. In this way, a DML script for creating a particular function may be stored in a subdirectory associated with the first version of the database. When modifications are made to the function to support a later release, the script file for the function may be copied to the subdirectory associated with the later release, and modified. In this way, the script containing the original version of the function is available if a user wanted to install the earlier version of the database, but the modified script is also available to support the newer version of the database.

**[0083]** In an alternate implementation, rather than organizing the scripts by subdirectory, the scripts may be managed using a source version control system (e.g., ClearCase, Source Safe, CVS, etc.). The source version control system may maintain multiple versions of a single script file, as the file is modified through multiple versions of the relational database. In this implementation, the XML file in which the database schema data is stored may also include a version number associated with each script file such that the version number corresponds to a version number maintained by the source version control system. This type of an implementation enables the creation of an installation file for any version of the relational database (current or previous), as well as the creation of an upgrade file from any previous version to any newer version.

**[0084]** At block 1006, the installation file is executed. For example, to install the initial version of the database on testing server 914, the installation file generated by upgrade/installation file generator 910 is copied to testing server 914.

The scripts in the installation file are then executed to create an instance of the initial version of the database. In an exemplary implementation, the scripts in the installation file are executed as a single transaction that can be committed or rolled back in its entirety. This prevents a partial install in an event that one of the DDL or DML scripts contains an error.

### **Non-Initial Version Database Installation Method**

[0085] Figure 11 is a flow diagram that illustrates an exemplary method 1100 for installing a non-initial version of a relational database. The illustrated process can be implemented in any suitable hardware, software, firmware or combination thereof.

[0086] At block 1102, the system identifies scripts associated with data definition language (DDL) objects that have been part of the database version to be installed or any previous version of the database. For example, upgrade/installation file generator 910 examines data in XML file 904 to identify files containing DDL scripts associated with the first version of the database, the second version of the database, the third version of the database, and so on up to the version of the database to be installed. An exemplary method for identifying the DDL scripts to be added to the installation file is illustrated in Figure 5b, and described above with reference to determination of set 504.

[0087] At block 1104, the system copies the DDL scripts that are identified into an installation file. For example, upgrade/installation file generator 910 extracts scripts from files indicated by the ddl elements identified in XML file 904, and copies the extracted scripts to an installation file.

[0088] At block 1106, the system identifies DML scripts associated with any version up to and including the version to be installed, not including DML scripts

for DML elements that have been dropped and not re-created between the first version and the version to be installed. For example, upgrade/installation file generator 910 iteratively steps through the data stored in XML file 904 by release, adding to a set DML scripts specified by sproc elements and removing from the set DML scripts specified by drop elements. An exemplary method for identify the DML scripts to be added to the installation file is illustrated in Figure 5c, and described above with reference to determination of set 506.

[0089] At block 1108, the system prepends a “create” command to the identified data manipulation language (DML) scripts, and copies the scripts into the installation file.

[0090] The installation file that is generated contains scripts to create the DDL and DML objects associated with the non-initial version of the database, and running the installation file will cause a new instance of the non-initial version of the database to be installed.

[0091] As described above, to support creation of an installation file associated with any version of the relational database, previous or current, script files may be organized in subdirectories that correspond to different versions of the database. Alternatively, a source version control system may be implemented to manage multiple versions of one or more script files.

[0092] At block 1110, the installation file is executed. For example, to install a non-initial version of the database on a production server 914, the installation file generated by upgrade/installation file generator 910 is copied to production server 916. The scripts in the installation file are then executed to create an instance of the non-initial version of the database. In an exemplary implementation, the scripts in the installation file are executed as a single

transaction that can be committed or rolled back in its entirety. This prevents a partial install in an event that one of the DDL or DML scripts contains an error.

### **Database Upgrade File Creation Method**

[0093] Figure 12 is a flow diagram that illustrates an exemplary method 1200 for upgrading a relational database. The illustrated process can be implemented in any suitable hardware, software, firmware or combination thereof.

[0094] At block 1202, the system identifies scripts associated with data definition language (DDL) objects that are associated with any version of the database after the version being upgraded from, up to and including the version being upgraded to. For example, upgrade/installation file generator 910 examines data in XML file 904 to identify files containing DDL scripts associated with the versions of the database greater than the version being upgraded from, up to and including the version being upgraded to. An exemplary method for identifying DDL scripts to be added to the upgrade file is illustrated in Figure 6b, and described above with reference to determination of set 604.

[0095] At block 1204, the system copies the identified DDL scripts into an upgrade file. For example, upgrade/installation file generator 910 extracts scripts from files indicated by the ddl elements identified in XML file 904, and copies the extracted scripts into an upgrade file.

[0096] At block 1206, the system identifies DML scripts associated with DML objects that have been created or modified between the version being upgraded from and the version being upgraded to. For example, upgrade/installation file generator 910 examines data in XML file 904 to identify files containing DML scripts associated with the versions of the database greater than the version being upgraded from, up to and including the version being upgraded to, not including

DML scripts for DML elements that have been dropped and not re-created between the version being upgraded from and the version being upgraded to. For example, upgrade/installation file generator 910 iteratively steps through the data stored in XML file 904 by release (beginning with the first version after the version being upgraded from), adding to a set, DML scripts specified by sproc elements and removing from the set DML scripts specified by drop elements. An exemplary method for identifying DML scripts associated with DML objects that have been added or modified is illustrated in Figure 7, and described above with reference to determination of set 620.

**[0097]** At block 1208, the system identifies DML scripts associated with DML objects that have been modified between the version being upgraded from and the version being upgraded to. For example, upgrade/installation file generator 910 first examines data in XML file 904 to identify DML objects that are associated with the version of the database that is being upgraded from. (This corresponds to set 618 shown in Figure 6c, and described above with reference thereto.) Upgrade/installation file generator 910 then performs an intersection between the set of DML scripts associated with DML objects present in the database version being upgraded from and the set of DML scripts associated with DML objects that are added or modified between the version being upgraded from and the version being upgraded to (the DML scripts identified as described above with reference to block 1206). An exemplary method for identifying DML scripts associated with DML objects that are modified is illustrated in Figures 6a – 7, and described above with reference thereto.

**[0098]** At block 1210, the system copies to an upgrade file, the DML scripts associated with DML objects that have been modified between the version being

upgraded from and the version being upgraded to (the DML scripts identified as described above with reference to block 1208). For example, upgrade/installation file generator 910 prepends an “alter” command to the identified data manipulation language (DML) scripts, and copies the scripts into the upgrade file.

**[0099]** At block 1212, the system identifies DML scripts that are associated with DML objects that have been newly created between the version being upgraded from and the version being upgraded to.

**[0100]** For example, as described above with reference to block 1206, upgrade/installation file generator 910 identifies a first set of DML scripts associated with DML objects that have been created or modified between the version being upgraded from and the version being upgraded to. An exemplary method for identifying this first set is illustrated in Figure 7, and described above with reference thereto.

**[0101]** Upgrade/installation file generator 910 then identifies a second set of DML scripts associated with DML objects that have been modified between the version being upgraded from and the version being upgraded to. As described above with reference to block 1208, an exemplary method for identifying this second set is illustrated in Figure 6c and is described above with reference thereto.

**[0102]** The system then determines a set of DML scripts associated with DML objects that have been newly created between the version being upgraded from and the version being upgraded to by finding the difference between the first set and the second set. This set is represented in Figure 6d as set 608, and is described above with reference thereto.

**[0103]** At block 1214, the system copies to an upgrade file, the DML scripts associated with DML objects that have been added between the version being

upgraded from and the version being upgraded to (the DML scripts identified as described above with reference to block 1212). For example, upgrade/installation file generator 910 prepends a “create” command to the identified data manipulation language (DML) scripts, and copies the scripts into the upgrade file.

**[0104]** In an alternate implementation, to ensure that new objects are created before existing objects are modified, the processing described with reference to block 1210 may actually be performed after the processing described with reference to block 1214.

**[0105]** At block 1216, the system identifies drop scripts that are associated with DML objects that have been dropped and not re-created between the version being upgraded from and the version being upgraded to. For example, as illustrated in Figure 6e, and described thereto with reference to set 610, a set is created of drop scripts associated with versions after the version being upgraded from, up to and including the version being upgraded to. (This set corresponds to the union of sets 622, 624, ..., 626, as illustrated in Figure 6e.) The set of drop scripts for the upgrade is identified as the difference between this set and the set of DML scripts associated with DML objects that have been added or modified between the version being upgraded from and the version being upgraded to (the set identified as described above with reference to block 1206, and represented as set 620 in Figure 6e).

**[0106]** At block 1218, the system copies to an upgrade file, the drop scripts associated with DML objects that have been dropped between the version being upgraded from and the version being upgraded to (the drop scripts identified as described above with reference to block 1216). For example, upgrade/installation file generator 910 prepends a “drop” command to the identified data manipulation

language (DML) scripts, and copies the scripts into the upgrade file. In an alternate implementation, the drop elements in the XML file specify only the name of the procedure or function to be dropped, rather than specifying the name of a script file, as illustrated in Figure 3. In such an implementation, upgrade/installation file generator 910 determines whether the specified name refers to a function or procedure, and generates a “drop procedure” or “drop function” call as appropriate.

**[0107]** As described above, to support creation of an upgrade file to upgrade any previous version of the database to any newer version of the database (even if the newer version is not the current version), script files may be organized in subdirectories that correspond to different versions of the database. Alternatively, a source version control system may be implemented to manage multiple versions of one or more script files.

**[0108]** At block 1220, the upgrade script is executed. For example, to upgrade a version installed on testing server 914, the upgrade file generated by upgrade/installation file generator 910 is copied to testing server 914. The scripts in the upgrade file are then executed to modify, create, and/or drop DDL and/or DML objects associated with the already installed version of the database to upgrade it to the new version of the database. In an exemplary implementation, the scripts in the upgrade file are executed as a single transaction that can be committed or rolled back in its entirety. This prevents a partial upgrade, which could be unstable, in an event that one of the DDL or DML scripts contains an error.

## **Conclusion**

[0109] The systems and methods described above enable relational database schema version management. Although the invention has been described in language specific to structural features and/or methodological steps, it is to be understood that the invention defined in the appended claims is not necessarily limited to the specific features or steps described. Rather, the specific features and steps are disclosed as preferred forms of implementing the claimed invention.